構成的型理論に基づいた定理証明プログラムの試作

A Prototype Program of Theorem Prover based on the Constructive Type Theory

石 井 忠 夫*

概 要

情報化社会においてソフトウェアの迅速で妥当な開発技術が強く求められているが，その要求に答えるにはソフトウェア開発方法の見直しが必要である。その１つの方法として，構成的プログラミングの考えに基づいてソフトウェアの仕様からその妥当性の検証とプログラムの導出を同時に実現し，更に，順次に追加される仕様の要求に対して，ソフトウェアを内部で矛盾を解消しながら拡大･発展させる見方がある。本稿では，このような枠組みの中で必要となる仕様からプログラムを導出するための定理証明プログラムを構成的型理論に基づいてRuby 言語を用いて試作した。

1　はじめに

現代の情報化社会においてはソフトウェアの迅速で妥当な開発技術が強く求められているが，また同時に他方では，過去に開発された多くのソフトウェア資産を有効に活用する枠組みも求められている。これらの要求に答えるためには，ソフトウェアの開発方法の見直しが必要である。通常のソフトウェア開発は対象領域の分析とモデル化からスタートし，どのような働きをするプログラムを必要とするのかを見極める必要がある。次に，このモデル化の結果を活用して，ソフトウェアの振る舞いを正しく規定した形式的言語による仕様書が作成される。一端仕様書が定まると，それを満たすプログラムを設計および実装し，また，テストや評価を通してプログラムが仕様を満たすことを確認する。ここで，仕様書が形式的言語で正しく矛盾なく記述されていれば，上述のプログラム言語への変換過程はいろんな課題はあると思われるが，しかし，原理的に自動化可能である。実際に，UML 仕様記述から Java コードへの変換も既に行われている。しかし，仕様の記述に誤りや矛盾が存在すると，そこから変換されたコードも誤りを含むことになる。更に，このような開発方法においては，矛盾を含まない仕様の記述が求められるので，仕様を後から順次追加した時に矛盾を見つけ出し，訂正や除去するような枠組みが必要となる。

このようなソフトウェアの開発方法は，「ソフトウェアが順次に追加される仕様の要求に対して，内部で矛盾を解消しながら拡大･発展すること」，と捉える見方であり，片山により最初に提案された[8]。この方法論においては，ソフトウェアは仕様の追加と共に矛盾を解消しながら順次発展するが，それと呼応して仕様を満たすプログラムも順次発展することが要求される。このようなソフトウェアの発展原理の確立には多くの課題が存在すると思われるが，仕様からその仕様を満たすプログラムを導出する枠組みの１つとして，R.L. Constable，後藤繁樹，佐藤雅彦らにより指摘され構成的プログラミングの手法がある。この基本的な考えは，直観的（または構成的）な推論を用いて論理式の証明を行う型の体系を定義する時，型が論理式（仕様）また型の対象が証明（仕様を満すプログラム）に対応することであり，この対応は Curry-Howard 同型対応と呼ばれる。この対応関係を満す型理論として既にいくつか提案されているが，その代表的なものの１つとして 1980 年代の初めに提案されたのが Martin-Löf の型理論 **MTT**［9，10］である。

先に，上述の構成的プログラミングの枠組みの中で片山が示した発展問題の形式化について検討を試みた[6, 7]。本稿では上の経緯を踏まえ，構成的プログラミングの枠組みを実現するために Martin-Löf の型理論 **MTT** に基づいた定理証明プログラムを試作した。既に，**MTT** に基づいた定理証明プロ

*ISHII, Tadao［情報システム学科］

グラムとして Agda2 や NuPRL [3] があり，また，Girard の構成的型理論 **F** に基づいた定理証明プログラムとして Coq [4] が知られている。Agda2 は Haskell 言語，また，NuPRL と Coq は ML 言語で記述されている。Haskell や ML は関数型言語であり，定理証明プログラムの記述に利用されるケースが多いが，本プログラムの試作ではオブジェクト指向型言語 Ruby を使用した。これは Ruby 言語の記述力が高く，多相型を含め関数言語の記述力を備え，また，自分自身を改変するリフレクションやメタプログラミングの記述が可能であり，ドメインに特化した固有言語として利用できるので上の目的に適っていると判断したためである[5]。本稿では，Ruby 言語による **MTT** に基づいた定理証明の実現について検討し，Π，$\sum$，＋型を対象に動作を確認した。

第 2 節では構成的型体系の 1 つとして Martin-Löf の型理論 **MTT** を概説する。第 3 節では **MTT** 内での定理の証明とサブゴールの表現について説明する。第 4 節では型と表現の構文定義，ゴールノードのデータ構造および実行コマンドの種類を説明し，本定理証明プログラムの概要を紹介する。第 5 節では本プログラムを用いた定理証明の具体例についていくつかを紹介する。第 6 節では今後の課題について議論する。

2　構成的型理論

構成的型理論はその上で構成的数学 [2] が展開可能な型理論であり，何かの存在を証明するためにはその対象を構成する具体的な手段を示す必要がある。そのため証明の対象に現れる論理結合子と量化子に対して，次に示す構成的な解釈が要請される。

⑴　矛盾⊥の証明 p は構成できない。

⑵　p が $A \wedge B$ の証明 $\Longleftrightarrow p = (q,r)$ であり，q が A の証明，かつ r が B の証明である。

⑶　p が $A \vee B$ の証明 $\Longleftrightarrow p = (n,q)$ であり，$n = 0$ なら q が A の証明，$n = 1$ なら q が B の証明である。

⑷　p が A → B の証明 $\Longleftrightarrow$ A の証明 q を B の証明に変換する手段 p がある。

⑸　p が$(\forall x)$A の証明 $\Longleftrightarrow$ 与えられた証明 t から $A\,[t/x]$ の証明に変換する手段 p がある。

⑹　p が$(\exists x)$A の証明 $\Longleftrightarrow$ 与えられた証明 t に対し $p = (t,q)$ であり，q が $A\,[t/x]$ の証明である。

この構成的な解釈を要請すると，次の 3 つの性質が成り立つ。特に，この (3°) の性質より，構成的証明からその仕様型を満たすプログラムを取り出すことができる。

(1°) $(\exists x)A$ が証明できると，$A(t)$ を満たす項 t が構成的に存在する。

(2°) $A \vee B$ が証明できると，A と B のどちらが証明されたかを判別できる。

(3°) $(\forall x)\,(\exists y)A(x,y)$が証明できると，$(\forall x)A(x,f(x))$ を満たすプログラム f を構成的に示せる。

次に，このような構成的型理論として Martin-Löf の型理論 **MTT** を取り上げる。**MTT** の形式体系は，表現，型，判定，および推論規則から構成される。表現とは型の対象であり，表現 a が型 A を持つことを $a \in A$ で表す。型に属する表現間の等号関係＝は次の評価規則により規定される。n 変数を持つ任意の表現 $b\,(x_1, x_2,..., x_n)$ において，各変数への別表現 $a_1, a_2,..., a_n$ の同時代入 $b\,[a_1, a_2,..., a$n/$x_1, x_2,..., x_n]$を表現 b の評価と定める。この時，各型に対して正規形表現と非正規形表現が定義できる。正規形表現は評価しても値が変らない形式であり定数データに対応し，また，非正規形表現は評価により値が変る形式でありプログラムデータに対応する。**MTT** は実際に表 2.1 に示す型と表現から構成される。判定は型理論における基本的な言明であり，**MTT** は⑴ A type，⑵ $A = B$，⑶ $a \in A$，⑷ $a = b \in A$ の 4 つで構成される。⑴ A は型であるや A は問題の仕様である，⑵ A と B は同じ型

であるや A と B は同じ問題である，(3) a は型 A の対象であるや a は問題 A のプログラムである，(4) a と b は型 A の等しい対象であるや a と b は問題 A の等しいプログラムである等に解釈できる。

表 2.1：MTT の型と表現

型　名　称	型　形　式	正規形表現	非正規形表現
有限集合（列挙型）	$N_n(n = 0, 1,...)$	$0_n, 1_n,..., (n-1)_n$	$R_n(c, c_0, c_1,..., c_{n-1})$
自然数（帰納型）	N	$0, succ(n)$	$R_c(c, d, e(x, y))$
集合族（依存積型）	$(\Pi x \in A)B(x)$	$(\lambda x)b$	$Ap(c, a)$
直積（レコード型）	$(\sum x \in A)B(x)$	$<a, b>$	$E_s(c, d(x, y))$
直和（バリアント型）	$A+B$	$inl(a), inr(b)$	$D_u(c, d(x), e(y))$
等号	$I(A, a, b)$	r	$J_g(c, d)$
リスト	$List(A)$	$nil, cons(a, b)$	$L_r(c, d, e(x, y, z))$
超限帰納型	$(Wx \in A)B(x)$	$sup(a, b)$	$T_A(c, d(x, y, z))$

MTT の推論規則はこれらの判定を用いて Gentzen の自然演繹体系で与えられる。各型に対して構成的解釈を自然な形で与えるように 4 つの推論規則が定義される。以下にリストおよび超限帰納型を除く型の推論規則を示す。

(1)　N_n 型

- N_n-形成規則：$\dfrac{}{N_n \text{ type}}$　$(n = 0, 1,...)$
- N_n-導入規則：$\dfrac{}{m_n \in N_n}$　$(m = 0, 1,..., n-1)$
- N_n-除去規則：$\dfrac{c \in N_n \quad c_m \in C(m_n) \quad \begin{matrix}[z \in N_n] \\ C(z) \text{ type}\end{matrix}}{R_n(c, c_0,..., c_{n-1}) \in C(c)}$　$(m = 0, 1,..., n-1)$
- N_n-等号規則：$\dfrac{\begin{matrix}c_m \in C(m_n) \\ \end{matrix} \quad \begin{matrix}[z \in N_n] \\ C(z) \text{ type}\end{matrix}}{R_n(m_n, c_0,..., c_{n-1}) = c_m \in C(c_m)}$　$(m = 0, 1,..., n-1)$

(2)　N 型

- N-形成規則：$\dfrac{}{N \text{ type}}$
- N-導入規則：$\dfrac{}{0 \in N} \qquad \dfrac{n \in N}{succ(n) \in N}$
- N-除去規則：$\dfrac{c \in N \quad d \in C(0) \quad \begin{matrix}[x \in N, y \in C(x)] \\ e(x, y) \in C(succ(x))\end{matrix} \quad \begin{matrix}[z \in N] \\ C(z) \text{ type}\end{matrix}}{R_c(c, d, e(x, y)) \in C(c)}$
- N-等号規則：$\dfrac{d \in C(0) \quad \begin{matrix}[x \in N, y \in C(x)] \\ e(x, y) \in C(succ(x))\end{matrix}}{R_c(0, d, e(x, y)) = d \in C(0)}$

$$\frac{n \in N \quad d \in C(0) \quad \overset{[x \in N,\ y \in C(x)]}{e(x,\ y) \in C(succ(x))}}{R_c(succ(n),\ d,\ e(x,\ y)) = e(n,\ R_c(n,\ d,\ e(x,\ y))) \in C(succ(n))}$$

(3) Π 型

● Π-形成規則：
$$\frac{\overset{[x \in A]}{A \text{ type} \quad B(x) \text{ type}}}{(\Pi x \in A)B(x) \text{ type}}$$

● Π-導入規則：
$$\frac{\overset{[x \in A]}{A \text{ type} \quad b(x) \in B(x)}}{(\lambda x)b(x) \in (\Pi x \in A)B(x)}$$

● Π-除去規則：
$$\frac{c \in (\Pi x \in A)B(x) \quad a \in A}{Ap(c,\ a) \in B(a)}$$

● Π-等号規則：
$$\frac{\overset{[x \in A]}{a \in A \quad b(x) \in B(x)}}{Ap((\lambda x)b(x),\ a) = b(a) \in B(a)}$$

(4) ∑型

●∑-形成規則：
$$\frac{\overset{[x \in A]}{A \text{ type} \quad B(x) \text{ type}}}{(\sum x \in A)B(x) \text{ type}}$$

●∑-導入規則：
$$\frac{\overset{[x \in A]}{B(x) \text{ type} \quad a \in A \quad b \in B(x)}}{<a,\ b> \in (\sum x \in A)B(x)}$$

●∑-除去規則：
$$\frac{c \in (\sum x \in A)B(x) \quad \overset{[x \in A,\ y \in B(x)]}{d(x,\ y) \in C(<x,\ y>)} \quad \overset{[z \in (\sum x \in A)B(x)]}{C(z) \text{ type}}}{E_s(c,\ d(x,\ y)) \in C(c)}$$

●∑-等号規則：
$$\frac{\overset{[x \in A]}{B(x) \text{ type} \quad a \in A \quad b \in B(a) \quad d(x,\ y) \in C(<x,\ y>)} \quad \overset{[x \in A,\ y \in B(x)]}{} \quad \overset{[z \in (\sum x \in A)B(x)]}{C(z) \text{ type}}}{E_s(<a,\ b>,\ d(x,\ y)) = d(a,\ b) \in C(<a,\ b>)}$$

(5) ＋型

● ＋-形成規則：
$$\frac{A \text{ type} \quad B \text{ type}}{A + B \text{ type}}$$

● ＋-導入規則：
$$\frac{a \in A}{inl(a) \in A + B} \qquad \frac{b \in B}{inr(b) \in A + B}$$

● ＋-除去規則：
$$\frac{c \in A + B \quad \overset{[x \in A]}{d(x) \in C(inl(x))} \quad \overset{[y \in B]}{e(y) \in C(inr(y))} \quad \overset{[z \in A + B]}{C(z) \text{ type}}}{D_u(c,\ d(x),\ e(y)) \in C(c)}$$

● ＋-等号規則：
$$\frac{a \in A \quad B \text{ type} \quad \overset{[x \in A]}{d(x) \in C(inl(x))} \quad \overset{[y \in B]}{e(y) \in C(inr(y))}}{D_u(inl(a),\ d(x),\ e(y)) = d(a) \in C(inl(a))}$$

$$\frac{A \text{ type} \quad b \in B \quad \overset{[x \in A]}{d(x) \in C(inl(x))} \quad \overset{[y \in B]}{e(y) \in C(inr(y))}}{D_u(inr(b),\ d(x),\ e(y)) = e(b) \in C(inr(b))}$$

⑹ I 型

- I-形成規則： $\dfrac{A\ \text{type}\quad a \in A\quad b \in A}{I(A, a, b)\ \text{type}}$
- I-導入規則： $\dfrac{a = b \in A}{r \in I(A, a, b)}$
- I-除去規則： $\dfrac{c \in I(A, a, b)}{a = b \in A}\qquad \dfrac{c \in I(A, a, b)\quad d \in C(r)}{J_g(r, d) \in C(c)}$
- I-等号規則： $\dfrac{c \in I(A, a, b)}{c = r \in I(A, a, b)}\qquad \dfrac{a = b \in A\quad d \in C(r)}{J_g(r, d) = d \in C(r)}$

3 定理証明とサブゴール表現

型理論を用いたプログラミングは形式体系での数学的定理の証明とみなせる。この時，プログラムの導出法として次の 2 つが考えられる。

⑴ ボトムアップ法

これはプログラムの小さな断片を最初に構成し，その断片を組み合わせて所望のプログラムを作る方法であり，これは公理と仮定から推論規則を繰り返し適用して論理的定理を証明する方法に当たる。

⑵ トップダウン法

元のプログラム仕様を段階的に詳細化しながら順次プログラムを作る方法であり，これは任意の仕様（ゴール）を部分仕様（サブゴール）に分割して証明し，その結果より元の証明を得る方法である。

Martin-Löf の型理論 **MTT** の証明に対し，上のトップダウン法の適用を考える。ゴールからサブゴールを得る方法は tactics と呼ばれるが [11]，**MTT** の tactics は各推論規則に対して下から上に読むことで得られる。例えば，Π-形成規則は次のサブゴールに分解できる。

01) $\downarrow (\Pi x \in A) B(x)$ type　（Π―形成規則）
02)　A type　　サブゴール 1
03)　$|[x \in A$　　仮定 1
04)　$|\triangleright\ B(x)$ type　　サブゴール 2
05)　$]|$
06) $\uparrow$ true または false

従って，上の推論規則は 01) の Π 型が成り立つためには，02) のサブゴール 1 及び 03) の仮定 1 の下で 04) のサブゴール 2 が成り立つことを確認すれば良い。また，次の Π-導入規則は次のサブゴールに分解できる。

01) $\downarrow (\Pi x \in A) B(x)$　（Π―導入規則）
02)　A type　　サブゴール 1
03)　$|[x \in A$　　仮定 1
04)　$|\triangleright\ \downarrow B(x)$　　サブゴール 2
05)　　$\uparrow \ni b(x)$
06)　$]|$
07) $\uparrow \ni (\lambda x) b(x)$

これは，02）のサブゴール 1 が成り立ち，更に 03）の仮定 1 の下で 04）のサブゴール 2 の対象 $b(x)$ が求まる（証明できる）時には，01）の Π 型の対象が $(\lambda x)b(x)$ で求まることを表してしる。この時，型 $(\Pi x \in A)(\Pi y \in B)A$ の証明は次で表せる。

01) $\downarrow(\Pi x \in A)(\Pi y \in B)A$（Π—導入規則）
02) A type　　サブゴール 1
03) $|[x \in A$　　仮定 1
04) $|\triangleright \downarrow(\Pi y \in B)A$　　サブゴール 2
05) B type　　サブゴール 3
06) $|[y \in B$　　仮定 2
07) $|\triangleright \downarrow A$　　サブゴール 4
08) $\uparrow \ni x$
09) $]|$
10) $\uparrow \ni (\lambda y)x$
11) $]|$
12) $\uparrow \ni (\lambda x)(\lambda y)x$

ここで，07）のサブゴール 4 は 03）の仮定 1 より求まり，これに Π-導入規則を 2 回適用することで，順次サブゴール 2，サブゴール 1 の対象が求まる。

4　定理証明プログラム

Martin-Löf の型理論 **MTT** に基づいた定理証明プログラムをオブジェクト指向スクリプト言語 Ruby を用いて作成した。以下では，本定理証明プログラムの概要について紹介する。

4.1　型と表現の定義

MTT の型と表現の構文を表 4.1 に示す。例えば，型 $(\Pi x \in A)(\Pi y \in B)A$ は，$<Pi, x : A, <Pi, y : B, A>>$ または $A->(B->A)$ と表現する。また，この型の対象である表現 $(\lambda x)(\lambda y)x$ は $\lambda[x](\lambda[y](x))$ と表現する。これらの型および表現の構文解析には Ruby で書かれたパーサジェネレータ Racc を使用した。

表 4.1：**MTT** の型と表現の構文

型　名　称	型　　構　　文	正規形構文	非正規形構文
有限集合（列挙型）	$<Nm,[a, b, c,...]>$ または[a, b, c,...]	?0, ?1,..., $?(n-1)$	$Rm(c, c_0, c_1,..., c_{n-1})$
自然数（帰納型）	$<Na>$ または Na	0, $succ(n)$	$Rc(c, d, \{x, y\} : e)$
集合族（依存積型）	$<Pi, x : A, B>$ または $A->B$	$\lambda[x](b)$	$Ap(c, a)$
直積（レコード型）	$<Sg, x : A, B>$ または $A \# B$	$pair(a, b)$	$Es(c, \{x, y\} : d))$
直和（バリアント型）	$<Ad, A, B>$ または $A \mid B$	$inl(a)$, $inr(b)$	$Du(c, x : d, y : e)$
等号	$<Id, A, [a, b]>$ または $a = [A]b$	r	$Jg(c, d)$
リスト	$<Lt, A>$ または $List(A)$	nil, $cons(a, b)$	$Lr(c, d, \{x, y, z\} : e)$
超限帰納型	$<Wo, x : A, B>$ または $A=>B$	$sup(a, b)$	$Tf(c, \{x, y, z\} : d)$

各型および表現は Ruby のクラスとして定義し，Racc のパーサからの出力として型または表現クラスのオブジェクトが返る。例えば，型構文$<Pi, x : A, <Pi, y : B, A>>$をパースすると，次のオブジェクトが生成される。

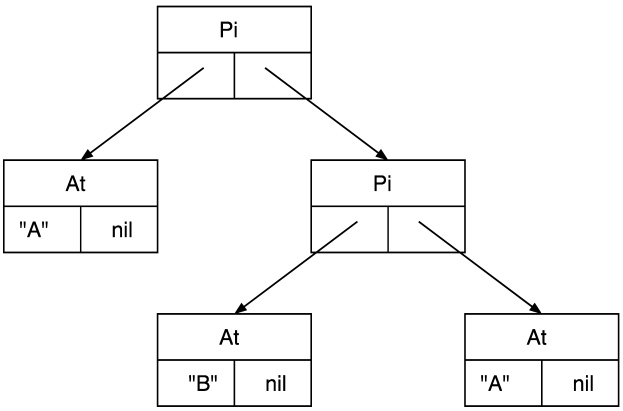

図 4.1：パースした型のオブジェクト

Pi，*Sg*，*Ad*，*Wo* などの 2 引数を伴う型構成子は，第一，第二引数型オブジェクトへのポインターを持つ。また，*At* は原子型（型変数）であり，第一引数はその名前，第二引数は他の型オブジェクトを代入可能である。

4.2 ゴールの定義

本節では，定理証明に現れるゴール／サブゴールの木構造について述べる。今，ゴール G が次のサブゴールに分解できるとする。

```
01) ↓G
02)   ↓G1              サブゴール 1
03)   ↓G2              サブゴール 2
04)  |[H1, H2          仮定 1
05)  |▷ ↓G3            サブゴール 3
06)     ↓G4            サブゴール 4
07)     |[H4, H5        仮定 2
08)     |▷ ↓G5          サブゴール 5
09)        ↓G6         サブゴール 6
10)           ↓G7      サブゴール 7
11)           −−+−−    OR サブゴール
12)           ↓G8      サブゴール 8
13)     ]|
14)  ]|
```

この時，ゴール G のサブゴール分解木は次で表現される。但し，$G6$ のサブゴールは $G7$ と $G8$ の OR サブゴールである。このように分解木の各ノードは**ゴール**，**仮定**または **OR ゴール**の 3 つに分類される。

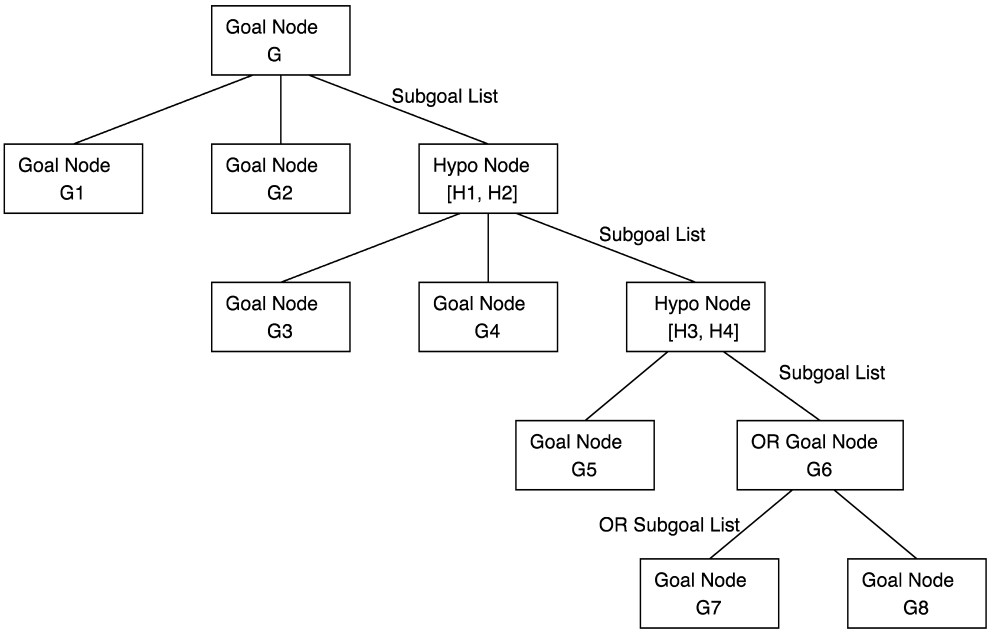

図 4.2：サブゴール分解木の構造

サブゴール分解木において，これら各ノードのデータ構造は次で構成される。

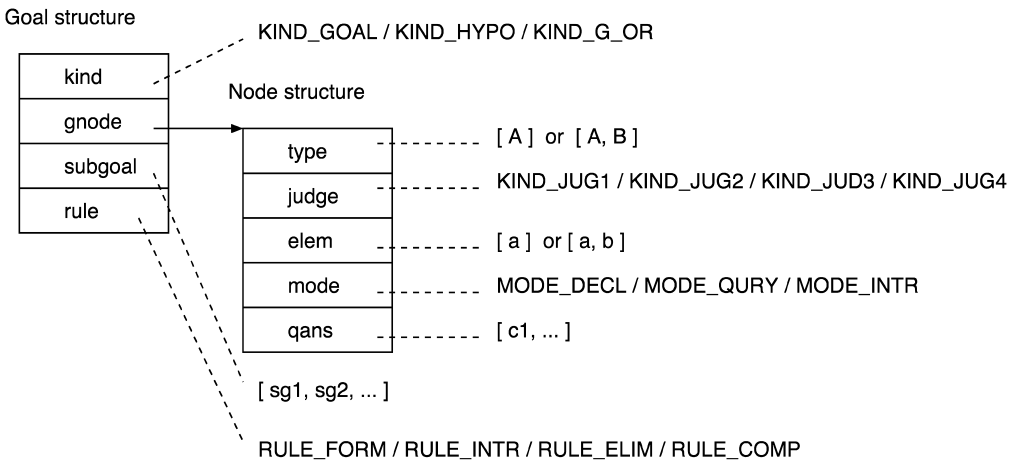

図 4.3：ゴールのデータ構造

ゴールのデータ構造において，kind はゴールの 3 分類，gnode はゴールデータでノードのデータ構造を持つ。subgoal はゴールの下のサブゴールリスト，rule はゴールへ適用する 4 種類の推論規則を表す。また，ノードのデータ構造において，type はノードの型オブジェクトリスト，judge はノードへ適用する 4 種類の判定（A type, $A = B$, $a \in A$, $a = b \in A$），elem は型の対象である表現リスト，mode は判定のアクションモード（判定の宣言，判定の問い合わせ，判定の導入），qans は判定の問い合わせ結果を格納する表現リストである。

4.3 実行コマンド

Windows のコマンド入力端末から本プログラムを起動すると，次の画面が表示される。コマンド操作は全て文字入力により行い，操作法は haskell インタープリタの Hugs を参考にしている。

```
D:\home\progm1\ruby\ruby-ctt\cttr>cttr-main.rb

 ====================================
   CCCC      tttttt   tttttt    RRRR
  C    C       tt       tt      R   R
 C             tt       tt      RRRR
 C                              R R
  C    C                        R  R
   CCCC                         R   R         Version: Dec 25 2009
 ------------------------------------------------------------------
 Constructive Type Theory with Ruby
 Type :? for help
CttR>
```

本プログラムはいくつかの操作コマンドを持つが，ヘルプ表示により確認できる。

```
CttR> :?
LIST OF COMMANDS:  Any command may be abbreviated to :d where
d is the first character in the full name.

:?                              display this list of commands
:def <spec name> <type name>    define specification name for the types
:undef <spec name>              undefine the specification name
:sub <nsp> <tsp[s1/t1,...]>     substitute spec name for type vars in spec name
:unsub <spec name>              unsubstitute the specification name
:list                           display the list of specification name
:show <spec name>               show type definition of the specification name
:load <file name>               load type definition from the specified files
:load                           clear all files except prelude
:edit <file name>               edit type definition for the specified files
:edit                           edit the last editing file
:goal <spec name>               display goal tree for the specification name
:proof <spec name>              proof types of the specification name
:redo <history number>          redo the specified history command
:redo                           redo the last doing command
:cd <dir>                       change the specified directory
:!<cmd>                         run the specified shell command
:history                        display the command history with serial numbers
:version                        print CttR version
:quit                           exit CttR interpreter
CttR>
```

主なコマンドについて簡単に説明する。: def コマンドは証明したい型に対して仕様名を定義する。: sub コマンドは指定した仕様名の型の中に現れる型変数に別の仕様名の型を代入する。: list コマンドは仕様名のリストを表示する。: show コマンドは仕様名とその型定義を表示する。: load コマンドはファイル内に指定されたコマンド操作を逐次実行する。: goal コマンドは仕様名の型に対するサブゴール分解木を表示する。: proof コマンドはサブゴール分解木を根から辿り，各サブゴールの証明を実行する。

5 具体例

ここでは定理証明の具体例をいくつか紹介する。

例 1　型 $A\rightarrow(B\rightarrow A)$　本仕様型のゴール分解木とその証明実行の結果を以下で示す。

```
--------------------==== Display of Goal Tree ====--------------------
 spec name : S1 = A->(B->A)
-----------------------------------------------------------------------
   1: ↓<Pi,A,<Pi,B,A>>
   2:    A type
   3:    |[ a1 ∈ A
   4:    |> ↓<Pi,B,A>
   5:          B type
   6:          |[ a2 ∈ B
   7:          |> ↓ A
   8:             ? ∋ a3
   9:          ]|
  10:       ? ∋ f2
  11:    ]|
  12: ? ∋ f1
------------------==============================------------------
```

ゴール分解木において，? はゴール型の対象が（構成的に）未導出であることを示している。$a3$ および $f1$ と $f2$ はゴール型の種別に応じて割り当てられた対象の変数名を表す。次に示すゴール型の証明実行により，これらの対象変数に導出された具体的表現が代入されると，ゴールが充足され ? が ↑ に変わる。また，表現 10）$\{a1\}:\lambda[a2](\{a2\}:a1)$ の中で，$a1$ は自由変数 ($\{a1\}$) また $a2$ は束縛変数 ([a2]) を表している。

```
---  proof execution of the spec name  ----
 spec name : S1 = A->(B->A)
 spec type :    = <Pi,A,<Pi,B,A>>
 spec objt : s1 = λ [a1]({a1}:λ [a2]({a2}:a1))

   1: ↓<Pi,A,<Pi,B,A>>
   2:    A type
   3:    |[ a1 ∈ A
   4:    |> ↓<Pi,B,A>
   5:          B type
   6:          |[ a2 ∈ B
   7:          |> ↓ A
   8:             ↑ ∋{a2}:a1
   9:          ]|
  10:       ↑ ∋{a1}:λ [a2]({a2}:a1)
  11:    ]|
  12: ↑ ∋ λ [a1]({a1}:λ [a2]({a2}:a1))
CttR>
```

上の証明実行において，07）のサブゴールは 03）の仮定より求まり，更に，04）と 01）のサブゴールは共に Π-導入規則の適用により求まる。

例 2　型 $A\wedge B\rightarrow A$　本仕様型の証明実行の結果を次に示す。

```
---  proof execution of the spec name  ----
 spec name : S2 = A#B->A
 spec type :    = <Pi,<Sg,A,B>,A>
 spec objt : s2 = λ [p1]({p1}:Es(p1,{a2,a3}:a2))

   1: ↓<Pi,<Sg,A,B>,A>
   2:    ↓<Sg,A,B> type
   3:       A type
   4:       B type
   5:    ↑ true
   6:    |[ p1 ∈<Sg,A,B>
   7:    |> ↓ A
   8:          |[ p2 ∈<Sg,A,B>
   9:          |> A type
  10:          ]|
  11:          |[ a2 ∈ A
  12:             a3 ∈ B
  13:          |> ↓ A
  14:             ↑ ∋{a2,a3}:a2
  15:          ]|
  16:       ↑ ∋{p1}:Es(p1,{a2,a3}:a2)
  17:    ]|
  18: ↑ ∋ λ [p1]({p1}:Es(p1,{a2,a3}:a2))
CttR>
```

上の証明実行において，最初に 07) のサブゴールを満たすことができないので 06) の仮定 $p1 \in <Sg, A, B>$ から$\sum$-除去規則を A を C とみなして適用する。これにより A のサブゴール 08)—15) が展開され，順次サブゴールの対象を導出すると 14)，16) を得る。最後に Π-導入規則を適用して 01) のゴール対象が求まる。

例 3　型 $A \to A \vee B$　本仕様型の証明実行の結果を次に示す。

```
---  proof execution of the spec name  ----
 spec name : S3 = A->A|B
 spec type :    = <Pi,A,<Ad,A,B>>
 spec objt : s3 = λ [a1]({a1}:inl(0,a1))

   1: ↓<Pi,A,<Ad,A,B>>
   2:    A type
   3:    |[ a1 ∈ A
   4:    |> ↓<Ad,A,B>
   5:          ↓ A
   6:          ↑ ∋ a1
   7:          --+--
   8:          ↓ B
   9:          ? ∋ a3
  10:       ↑ ∋{a1}:inl(0,a1)
  11:    ]|
  12: ↑ ∋ λ [a1]({a1}:inl(0,a1))
CttR>
```

この証明では，04) のサブゴールにおいて＋-導入規則を適用してその下のサブゴールを展開している。この時，2 つのサブゴール 05) および 08) はいずれか片方を導出すれば良いので OR-ゴールとなる。03) の仮定より，05) のサブゴールが求まり，04) のサブゴール対象は $a1 : inl(0, a1)$ と導出される。最後に Π-導入規則を適用して 01) のゴール対象が求まる。

例 4　型 $(A \to B) \to ((B \to C) \to (A \to C))$　本仕様型の証明実行の結果を次に示す。

```
---  proof execution of the spec name  ----
 spec name : S4 = (A->B)->((B->C)->(A->C))
 spec type :    = <Pi,<Pi,A,B>,<Pi,<Pi,B,C>,<Pi,A,C>>>
 spec objt : s4 = λ [f2]({f2}:λ [f4]({f4}:λ [a1]({a1}:Ap(f4,Ap(f2,a1)))))

  1: ↓<Pi,<Pi,A,B>,<Pi,<Pi,B,C>,<Pi,A,C>>>
  2:    ↓<Pi,A,B> type
  3:       A type
  4:       B type
  5:    ↑ true
  6:    |[ f2 ∈<Pi,A,B>
  7:    |> ↓<Pi,<Pi,B,C>,<Pi,A,C>>
  8:          ↓<Pi,B,C> type
  9:             B type
 10:             C type
 11:          ↑ true
 12:          |[ f4 ∈<Pi,B,C>
 13:          |> ↓<Pi,A,C>
 14:                A type
 15:                |[ a1 ∈ A
 16:                |> ↓ C
 17:                       ↓ B
 18:                          ↓ A
 19:                          ↑ ∋ a1
 20:                       ↑ ∋ Ap(f2,a1)
 21:                    ↑ ∋{a1}:Ap(f4,Ap(f2,a1))
 22:                ]|
 23:             ↑ ∋{f4}:λ [a1]({a1}:Ap(f4,Ap(f2,a1)))
 24:          ]|
 25:       ↑ ∋{f2}:λ [f4]({f4}:λ [a1]({a1}:Ap(f4,Ap(f2,a1))))
 26:    ]|
 27: ↑ ∋ λ [f2]({f2}:λ [f4]({f4}:λ [a1]({a1}:Ap(f4,Ap(f2,a1)))))
CttR>
```

最初に 16) のサブゴールの証明に失敗するので，次に仮定リストを検索して，Π-除去規則を適用すると 16) の C を導出するような Π 型の第一引数型（今の場合は B）をサブゴールに選択する。同様にして，B のサブゴールとして A が選択される。最後に，Π-除去規則および Π-導入規則を適用するとゴール対象が求まる。

例 5　型 $(A \to C) \vee (B \to C) \to (A \wedge B \to C)$)　本仕様型の証明実行の結果を次に示す。

```
---  proof execution of the spec name  ----
 spec name : S5 = (A->C)|(B->C)->(A#B->C)
 spec type :    = <Pi,<Ad,<Pi,A,C>,<Pi,B,C>>,<Pi,<Sg,A,B>,C>>
 spec objt : s5 = λ [d1]({d1}:λ [p1]({p1}:Es(p1,{a2,a3}:Du(d1,{f3}:Ap(f3,a2),{f4}:Ap(f4,a3)))))

  1: ↓<Pi,<Ad,<Pi,A,C>,<Pi,B,C>>,<Pi,<Sg,A,B>,C>>
  2:    ↓<Ad,<Pi,A,C>,<Pi,B,C>> type
  3:       ↓<Pi,A,C> type
  4:          A type
  5:          C type
  6:       ↑ true
  7:       ↓<Pi,B,C> type
  8:          B type
  9:          C type
 10:       ↑ true
 11:    ↑ true
 12:    |[ d1 ∈<Ad,<Pi,A,C>,<Pi,B,C>>
 13:    |> ↓<Pi,<Sg,A,B>,C>
 14:          ↓<Sg,A,B> type
 15:             A type
 16:             B type
 17:          ↑ true
 18:          |[ p1 ∈<Sg,A,B>
 19:          |> ↓ C
 20:                |[ p2 ∈<Sg,A,B>
 21:                |> C type
 22:                ]|
 23:                |[ a2 ∈ A
 24:                   a3 ∈ B
 25:                |> ↓ C
 26:                      |[ d2 ∈<Ad,<Pi,A,C>,<Pi,B,C>>
 27:                      |> C type
 28:                      ]|
 29:                      |[ f3 ∈<Pi,A,C>
 30:                      |> ↓ C
 31:                            ↓ A
 32:                            ↑ ∋ a2
 33:                         ↑ ∋{f3}:Ap(f3,a2)
 34:                      ]|
 35:                      |[ f4 ∈<Pi,B,C>
 36:                      |> ↓ C
 37:                            ↓ B
 38:                            ↑ ∋ a3
 39:                         ↑ ∋{f4}:Ap(f4,a3)
 40:                      ]|
 41:                   ↑ ∋{a2,a3}:Du(d1,{f3}:Ap(f3,a2),{f4}:Ap(f4,a3))
 42:                ]|
 43:             ↑ ∋{p1}:Es(p1,{a2,a3}:Du(d1,{f3}:Ap(f3,a2),{f4}:Ap(f4,a3)))
 44:          ]|
 45:       ↑ ∋{d1}:λ [p1]({p1}:Es(p1,{a2,a3}:Du(d1,{f3}:Ap(f3,a2),{f4}:Ap(f4,a3))))
 46:    ]|
 47: ↑ ∋ λ [d1]({d1}:λ [p1]({p1}:Es(p1,{a2,a3}:Du(d1,{f3}:Ap(f3,a2),{f4}:Ap(f4,a3)))))
CttR>
```

最初に 19) のサブゴールの証明に失敗するので，次に 18) の仮定 $p1 \in <Sg, A, B>$ から$\sum$-除去規則を適用する。これにより C のサブゴール 20)—25) が展開される。更にこのサブゴールの中で 25) が失敗するので，仮定リストを検索して C を部分型に持つ仮定（今の場合は 12 行目の $d1 \in <Ad, <Pi, A, C>, <Pi, B, C>>$）を選び，この仮定の下で除去規則（今の場合は＋-除去規則）を適用する。この結果として C のサブゴール 26)—40) が更に展開される。これらのサブゴールを導出し，＋-除去規則を適用した結果として 25) のサブゴール対象が求まり，更に$\sum$-除去規則を適用した結果として

19) のサブゴール対象が求まる。最後に，Π-導入規則を 2 回適用するとゴール対象が求まる。

6　今後の課題

本稿では，Martin-Löf の構成的型理論に基づいた定理証明プログラムを Ruby 言語を用いて試作した。現状は試作の段階であり，**MTT** が持つ基本型と型構成子を対象としている。結果としては Π，∑，＋型を対象に導入規則や除去規則を適用して証明の対象を構成できることを確認した。今後の課題としては次を考えている。

⑴　型の否定¬ A は A→⊥と定義されるが，このような型の否定を導入する。

⑵　**MTT** では最初から型全体の世界 **U** が存在するとは仮定せずに，基本型と型構成子から成る下層の型の集まり $\mathbf{U}_0$から順次に $\mathbf{U}_1$，$\mathbf{U}_2$...と構成的に定義しているが，このような型の世界の階層を導入する。

⑶　部分集合型（Subset），商集合型（Quotient），多重集合型（Bags）等の型を導入する［1］。

⑷　型の証明対象はプログラムと見なせるが，これを現代的なプログラム言語（Ruby や Java など）へのコード変換を導入する。

⑸　ソフトウェア仕様を型で直接表現するのは困難であるので，UML 等の仕様記述言語から型表現への変換を導入する。

参考文献

［1］ R. Backhouse, P. Chisholm, G. Malcolm and E. Saaman, *Do-it-yourself type theory*, Formal Aspects of Computing, vol. 1 (1989), pp. 19-84.

［2］ M. Beeson, *Foundations of Constructive Mathematics*, Springer-Verlag, 1985.

［3］ R.L. Constable et al., *Implementing Mathematics with the NuPRL Proof Development System*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

［4］ T. Coquand and G. Huet, *The Calculus of Constructions*, Information and Computation, vol. 76 (1988), pp. 95-120.

［5］ D. Flanagan and Y. Matsumoto, *The Ruby Programming Language*, O'Reilly Media, Inc, 2008.

［6］ 石井忠夫，ソフトウェア仕様の差分について，新潟国際情報大学情報文化学部紀要，vol. 10 (2007), pp. 147-154.

［7］ 石井忠夫，ソフトウェア仕様とプログラムの導出新潟国際情報大学情報文化学部紀要，vol. 12 (2009), pp. 141-150.

［8］ 片山卓也，ソフトウェア発展原理と研究課題，日本ソフトウェア科学会第 14 回大会論文集，(1997), pp. 297-300.

［9］ P. Martin-Löf, Constructive Mathematics and Computer Programming, in *Logic, Methodolgy and Philosophy of science VI*, eds. by L.J. Cohen et al., North-Holland, Amsterdam, 1982, pp. 153-179.

［10］ P. Martin-Löf, *Intuitionistic Type Theory*, Notes by Giovanni Sambin of a Series of Lectures Given in Padua, June 1980, Bibliopolis, Napoli, 1984.

［11］ T. Bengt, P. Kent and J.P. Smith, *Programming in Martin-Löf's Type Theory, An Introduction*, Oxford Science Publications, 1990.